Requested Patent: JP2001051965A

Title: ;

Abstracted Patent: JP2001051965 ;

Publication Date: 2001-02-23 ;

Inventor(s): ;

Applicant(s): ;

Application Number: JP19990228195 19990812 ;

Priority Number(s): JP19990228195 19990812 ;

IPC Classification:

G06F15/177 ; G06F15/177 ; G06F11/22 ; G06F12/08 ; G06F12/12 ; G06F12/16 ; G06F13/00 ;

Equivalents:

ABSTRACT:

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2001－51965
（P2001－51965A）
(43)公開日　平成13年２月23日(2001.2.23)

(51)Int.Cl.'		識別記号	FI		テーマコード'(参考)
Ｇ０６Ｆ	15/177	６７８	Ｇ０６Ｆ 15/177	６７８Ｈ	５Ｂ００５
		６８２		６８２Ｊ	５Ｂ０１８
	11/22	３５０	11/22	３５０Ｄ	５Ｂ０４５
	12/08		12/08	Ｓ	５Ｂ０４８
		３１０		３１０Ｂ	５Ｂ０８３

審査請求　未請求　請求項の数５　ＯＬ　（全 21 頁）　最終頁に続く

(21)出願番号　特願平11－228195

(22)出願日　平成11年８月12日(1999.8.12)

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号

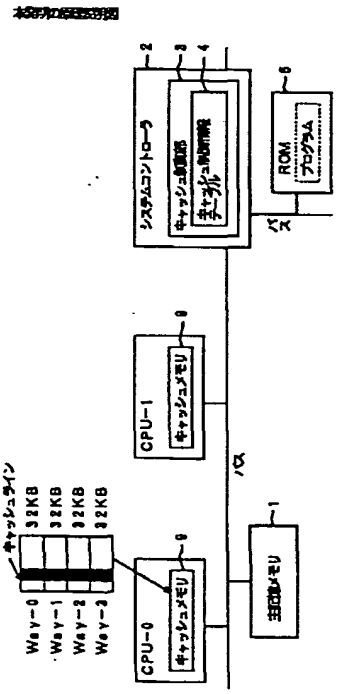
(72)発明者　市川　文男
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

(74)代理人　100096530
弁理士　今村　辰夫　（外２名）

最終頁に続く

(54)【発明の名称】　情報処理装置の試験方法及び記録媒体

(57)【要約】
【課題】本発明は情報処理装置の試験方法に関し、マルチＣＰＵを対象としたランダム命令試験で、各ＣＰＵが同一ブロックをアクセスでき、かつ別のＣＰＵからデータが破壊されない状態でキャッシュコヒーレンシィ試験を実現する。
【解決手段】複数のＣＰＵ（ＣＰＵ－０、ＣＰＵ－１・・・）をバスで接続し、各ＣＰＵにキャッシュメモリ９を備えた装置を対象としてキャッシュコヒーレンシィ試験を行う試験において、ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各ＣＰＵ毎に一定値づつずらした値に設定し、オペランドアドレスのインデックス部を指すインデックスレジスタの値と、オペランドアドレス内のディスプレースメント値を予め規定しておくことにより、全ＣＰＵに同一アクセス空間を割り当てた時の各ＣＰＵからの重複アクセスを回避するようにした。



【特許請求の範囲】
【請求項１】複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内で、ベースレジスタの値に、インデックスレジスタの値、或いはディスプレースメント値を加算して、アクセス命令のオペランドアドレスを生成することにより、ランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、
前記ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各ＣＰＵ毎に一定値づつずらした値に設定し、
前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておくことにより、全ＣＰＵに同一アクセス空間を割り当てた時の各ＣＰＵからの重複アクセスを回避することを特徴とする情報処理装置の試験方法。
【請求項２】複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、
同一論理空間を持ち、命令により切り換え可能な異なるコンテキストを複数用意し、該コンテキスト毎に、それぞれ論理アドレスと実メモリの物理アドレスとの対応情報を持ち、
前記ランダム命令列を実行させる過程で、特定命令により前記コンテキストを切り換えた際、同一論理アドレスをアクセスしても、実メモリの異なる物理アドレスを指すようにしてキャッシュリプレースを行わせ、キャッシュリプレース回数をより多く生成させることを特徴とする情報処理装置の試験方法。
【請求項３】複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、
各ＣＰＵが実行する前記ランダム命令列内に試験対象キャッシュアドレスの選択を行う特定命令を設けておき、この特定命令の実行により、各ＣＰＵとも、同一の試験対象キャッシュアドレスを選択し、このアドレスに、各ＣＰＵに割り振ったシフトバイトを加算した値を求め、この値により、各ＣＰＵから見て、同一キャッシュアドレスで、かつ、アクセスが重複しないようにしたことを特徴とする情報処理装置の試験方法。
【請求項４】複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、
生成したランダム命令列を別の空間に複写する処理と、
両方のランダム命令空間に対して、論理アドレスが等しく、物理アドレスが異なるアドレス変換テーブルを作成し、該テーブルに異なるコンテキスト値を定義する処理と、
初期値として、コンテキストに値を設定し、或る論理アドレスから試験を開始させる処理と、
或る割り込みを契機に、命令の論理アドレスを前記アドレス変換テーブルから選択し、この値にランダム命令内アドレスを命令カウンタに設定する処理と、
或る割り込みを契機に、コンテキストの値を切り換えることにより、論理アドレスの命令カウンタは変わらずに、物理アドレスのみ変更できるようにする処理とを有し、
前記処理により、ランダム命令列を頭から順次実行し、かつ実行空間のみ変化させることで、命令キャッシュのキャッシュコヒーレンシィ試験を可能にしたことを特徴とする情報処理装置の試験方法。
【請求項５】複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内で、ベースレジスタの値に、インデックスレジスタの値、或いはディスプレースメント値を加算して、アクセス命令のオペランドアドレスを生成することにより、ランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置に、
前記ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各ＣＰＵ毎に一定値づつずらした値に設定する手順と、
前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておく手順と、
を実行させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵシステムを対象としたランダム命令試験で、キャッシュコヒーレンシィ機能の検証を可能にした情報処理装置の試験方法及び記録媒体に関する。
【０００２】
【従来の技術】以下、従来例を説明する。
【０００３】§１：キャッシュコヒーレンシィの説明
近年、コンピュータの高速化の一手段として、主記憶メモリ（主記憶装置）とは別に、ＣＰＵにキャッシュメモリと呼ばれる数キロバイトから数メガバイトにも及ぶ高速メモリを配置し処理のスピードアップを図っている。この場合、主記憶メモリからキャッシュに読み込まれたデータは、キャッシュメモリ上で動作している間、常に

高速でデータ処理が行える。

【0004】この制御は、ＣＰＵ1台の時は自ＣＰＵ内部のキャッシュメモリと主記憶メモリとの一致性（Cache Coherency ：キャッシュコヒーレンシィ）を考慮するだけで済むが、複数のＣＰＵが搭載された場合は、ＣＰＵ相互のキャッシュメモリをも考慮する必要があり、複雑なハード制御となる。この制御を、図に基づいて簡単に説明すると次のようになる。

【0005】§2：システム構成と処理概要の説明・・・図15参照

図15は従来例のシステム構成図である。このシステムは、マルチＣＰＵシステムの1例であり、複数のＣＰＵ（ＣＰＵ－0、ＣＰＵ－1・・・）と主記憶メモリ1（全ＣＰＵで共有する1個のメモリ）と、システムコントローラ2等がバスで接続されている。そして、各ＣＰＵには、それぞれキャッシュメモリ（以下、単に「キャッシュ」とも記す）を持っている。

【0006】この場合、各ＣＰＵ内部には一次キャッシュ（以下「Ｌ1＄」と記す）を設け、各ＣＰＵのチップの外側には二次キャッシュ（以下「Ｌ2＄」と記す）を設けている。前記Ｌ1＄は小容量のキャッシュであり、Ｌ2＄はＬ1＄よりも大容量のキャッシュである。

【0007】また、前記バスにはシステムコントローラ2が接続され、該システムコントローラ2に接続された別のバスには入出力制御部（以下「Ｉ／Ｏ制御部」と記す）5を介してＲＯＭ6等が接続されている。前記システムコントローラ2は、システム内のキャッシュメモリの管理等を含む各種制御を行うものであり、該システムコントローラ2内に、キャッシュ情報テーブル4を有するキャッシュ制御部3が設けてある。

【0008】前記キャッシュ制御部3は、前記各ＣＰＵが持つＬ1＄、Ｌ2＄の情報をキャッシュ情報テーブル4に格納し、これらＬ1＄、Ｌ2＄のキャッシュ制御を行う機能を備えている。例えば、ＣＰＵが主記憶メモリ1のデータを取り込む場合、先ず、内部のＬ1＄を見にゆく。そして、Ｌ1＄に該当するデータがなければ、他のＣＰＵへ該当するデータを取りに行くが、この時、前記ＣＰＵはシステムコントローラ2へキャッシュ情報を貰いに行き、その情報に基づいて他のＣＰＵのデータ読み込む。

【0009】従って、システムコントローラ2は、各ＣＰＵのＬ1＄、Ｌ2＄の情報を、キャッシュ情報テーブル4に格納し、常に更新しながら管理しておき、各ＣＰＵに対してこの情報によりキャッシュ制御を行うようになっている。

【0010】前記システムにおいて、各ＣＰＵがロード命令（Load）、及びストア命令（Store ）を実行する場合の処理概要は次の通りである。例えば、前記ロード命令は、主記憶メモリ1のデータをＣＰＵ内のレジスタに取り込む処理であるが、この場合、最初のロード命令であれば、データは主記憶メモリ1→Ｌ2＄→Ｌ1＄→ＣＰＵ内部のレジスタの順に転送される。

【0011】また、ストア命令は、ＣＰＵ内のレジスタに格納されているデータを主記憶メモリ1に書き込む処理であるが、この命令を実行した場合の処理は次の通りである。先ず、ＣＰＵ内のレジスタに格納されているストアすべきデータに対応するデータがＬ1＄に有るかどうかを判断し、該データが有れば、Ｌ1＄の対応するデータを新データ（前記ストア対象のデータ）で書き換えて処理を終了する。この時、主記憶メモリ1のデータは書き換えないため古いデータのままである。

【0012】しかし、Ｌ1＄に前記ストア対象のデータに対応するデータが無ければ、ストアすべきデータに対応するデータがＬ2＄に有るかどうかを判断し、該データが有れば、Ｌ2＄の対応するデータを新データ（前記ストア対象のデータ）で書き換えて処理を終了する。この時、主記憶メモリ1のデータは書き換えないため古いデータのままである。

【0013】更に、前記ストアすべきデータに対応するデータが、Ｌ1＄にもＬ2＄にも無ければ、前記対応するデータを主記憶メモリ1から読み出してＬ2＄に書き込み、Ｌ2＄の対応するデータを新データ（前記ストア対象のデータ）で書き換えて処理を終了する。この時、主記憶メモリ1のデータは書き換えないため古いデータのままである。

【0014】なお、主記憶メモリ1にデータを書き込むのは、Ｌ2＄を追い出されたデータである。そして、前記の各処理はシステムコントローラ2が常に監視していて、そのキャッシュメモリの情報は、キャッシュ制御部3がキャッシュ情報テーブル4に格納している。また、Ｌ1＄に有るデータは、必ずＬ2＄にも有る。従って、各ＣＰＵのアクセス命令実行時には、システムコントローラ2からの前記キャッシュ情報テーブル4の情報を基に処理を行う。

【0015】なお、前記キャッシュの制御に関しては、「ＭＯＥＳＩ」の理論として多くの文献に記載されており、良く知られている（周知の理論）ので、詳細な説明は省略する。前記「ＭＯＥＳＩ」に関する参考文献の1例としては、例えば、次の参考文献が知られている。

【0016】参考文献：「ultraSPARC™-IUser´s Manual」，Revision 1.0,Sept 18,1995,「SPARC Technology Business」A Sun Microsystems,Inc.Business 2550 Garcia Avenue,Mountain View,CA 94043 U.S.A,Part No;STP1030-UG.(P91-97参照）.

§3：システム内の処理の説明

前記システムにおいて、例えば、ＣＰＵ－0が主記憶メモリ1の１００番地の内容をロード（Load）すると、ＣＰＵ－0のキャッシュ（Ｌ1＄）には主記憶メモリ1の１００番地から1ブロック（例えば、６４バイト）分のデータが読み込まれる。次に、主記憶メモリ1の１００

番地に新しいデータを格納（Store ）すると、ＣＰＵ－０のキャッシュ（Ｌ１＄、又はＬ２＄）は新しいデータに置き変わるが、主記憶メモリ１は古いデータを保持している。

【００１７】この状態でＣＰＵ－１が主記憶メモリ１の１００番地の内容をロードした場合、その時のデータは主記憶メモリ１から読み出されずにＣＰＵ－０のキャッシュ（Ｌ１＄、又はＬ２＄）から読み出される。これは最新の１００番地のデータをＣＰＵ－０のキャッシュ（Ｌ１＄、又はＬ２＄）が保持しているからである。このような機構を持つシステムにおけるキャッシュの一致性を検証する手法について次に述べる。

【００１８】§４：キャッシュの一致性を検証する手法の説明・・・図１６参照

図１６は、従来のランダム命令試験例である。一般的に、或る１つのタスクを複数同時に動作させる場合、スタック域、データアクセス域等のデータを書き換える領域をタスク数分設け、各タスクはそれぞれ自分に割り当てられた空間を使用することにより実現させている。この手法は、１つのタスクを複数のＣＰＵから共有して動作させる場合に当てはまり、一般的にはマルチＣＰＵのランダム命令試験もこの手法を用いている場合が多い。

【００１９】従来技術では、１台のＣＰＵ配下で動作しているランダム命令テスト（この中にはメモリをアクセスする命令も含まれている）を単に、複数のＣＰＵ配下で同時に動作させているだけである。つまり、ランダム命令が同時に複数ＣＰＵから実行されれば、そのランダム命令列中に含まれるアクセス命令も複数ＣＰＵからランダムに発信され、結果的にキャッシュ・主記憶メモリ間のテストができるという考えである。

【００２０】例えば、図１６に示した例では、ランダム命令列が異なり、メモリアクセス域が異なる第１のテスト例（図１６のA図参照）と、ランダム命令列が同じで、メモリアクセス域が異なる第２のテスト例（図１６のB図参照）を示している。この場合、前記テスト例１では、ＣＰＵ－０とＣＰＵ－１が、異なるメモリ域を使用し、異なる命令列を実行することでテストを行う。また、前記テスト例２では、ＣＰＵ－０とＣＰＵ－１が、同じメモリ域を使用し、異なる命令列を実行することで試験を行う。

【００２１】ここで問題となるのが、各ＣＰＵからのアクセス域が別空間に存在するため、或るＣＰＵのメモリアクセス動作が他のＣＰＵのキャッシュに対して作用することがない。すなわち、複数ＣＰＵに跨がってキャッシュコヒーレンシィテストができていないのが現状である。このテストを実現するためには、同一キャッシュラインに対してのアクセスが必要になるが、このような手法は提供されていないのが現状である。

【００２２】

【発明が解決しようとする課題】前記のような従来のものにおいては、次のような課題があった。

【００２３】前記のように、同一キャッシュライン（例えば、６４バイト）に対してのアクセスを可能にするためには、各ＣＰＵが同一ブロックアドレス（６４バイトバウンダリから６４バイト）をアクセスする必要がある。ここで、各ＣＰＵがアクセスする空間（メモリ領域）を同じにすれば良いと考えるかもしれないが、各ＣＰＵは非同期に動作しているため、各ＣＰＵが同一アドレスを使うとなると、タイミングによってメモリに書き込んだ内容を読み出した時に異なるデータを読み出すことがある。これは、そのアドレスの内容を読み出す直前に、別のＣＰＵがそのアドレスに別の内容を書き込む可能性があるからである。

【００２４】本発明は、このような従来の課題を解決し、マルチＣＰＵを対象としたランダム命令試験で、各ＣＰＵが同一ブロックをアクセスでき、かつ別のＣＰＵからデータが破壊されない手法を用いてキャッシュコヒーレンシィ試験を実現できるようにすることを目的とする。

【００２５】

【課題を解決するための手段】図１は本発明の原理説明図であり、１は主記憶メモリ、２はシステムコントローラ、３はキャッシュ制御部、４はキャッシュ制御情報テーブル、６はプログラムを格納したＲＯＭ、９はキャッシュメモリを示す。本発明は前記の目的を達成するため、次のように構成した。

【００２６】(1) ：複数のＣＰＵ（ＣＰＵ－０、ＣＰＵ－１・・・）をバスで接続し、各ＣＰＵにキャッシュメモリ９を備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内で、ベースレジスタの値に、インデックスレジスタの値、或いはディスプレースメント値を加算して、アクセス命令のオペランドアドレスを生成することにより、ランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、前記ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各ＣＰＵ毎に一定値づつずらした値に設定し、前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておくことにより、全ＣＰＵに同一アクセス空間を割り当てた時の各ＣＰＵからの重複アクセスを回避するようにした。

【００２７】(2) ：複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、同一論理空間を持ち、命令により切り換え可能な異なるコンテキストを複数用意し、該コンテキスト毎に、それぞれ論理アドレスと実メモリの物理アドレスとの対応情報を持ち、前記ランダム

命令列を実行させる過程で、特定命令により前記コンテキストを切り換えた際、同一論理アドレスをアクセスしても、実メモリの異なる物理アドレスを指すようにしてキャッシュリプレースを行わせ、キャッシュリプレース回数をより多く生成させるようにした。

【００２８】(3)：複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリ９を備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、各ＣＰＵが実行する前記ランダム命令列内に試験対象キャッシュアドレスの選択を行う特定命令を設けておき、この特定命令の実行により、各ＣＰＵとも、同一の試験対象キャッシュアドレスを選択し、このアドレスに、各ＣＰＵに割り振ったシフトバイトを加算した値を求め、この値により、各ＣＰＵから見て、同一キャッシュアドレスで、かつ、アクセスが重複しないようにした。

【００２９】(4)：複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリ９を備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、生成したランダム命令列を別の空間に複写する処理と、両方のランダム命令空間に対して、論理アドレスが等しく、物理アドレスが異なるアドレス変換テーブルを作成し、該テーブルに異なるコンテキスト値を定義する処理と、初期値として、コンテキストに値を設定し、或る論理アドレスから試験を開始させる処理と、或る割り込みを契機に、命令の論理アドレスを前記アドレス変換テーブルから選択し、この値にランダム命令内アドレスを命令カウンタに設定する処理と、或る割り込みを契機に、コンテキストの値を切り換えることにより、論理アドレスの命令カウンタは変わらずに、物理アドレスのみ変更できるようにする処理とを有し、前記処理により、ランダム命令列を頭から順次実行し、かつ実行空間のみ変化させることで、命令キャッシュのキャッシュコヒーレンシィ試験を可能にした。

【００３０】(5)：複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内で、ベースレジスタの値に、インデックスレジスタの値、或いはディスプレースメント値を加算して、アクセス命令のオペランドアドレスを生成することにより、ランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置に、前記ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各ＣＰＵ毎に一定値づつずらした値に設定する手順と、前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておく手順と、を実行させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。

【００３１】（作用）前記構成に基づく本発明の作用を、図１に基づいて説明する。

【００３２】(a)：前記(1)では、前記情報処理装置の試験を行う際、ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各ＣＰＵ毎に一定値づつずらした値に設定し、前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておく。

【００３３】このようにすれば、全てのＣＰＵから同一命令を実行した時に、アクセス命令のアクセスアドレスは、ＣＰＵ毎に必ず決められたアドレス範囲を指すことができる。すなわち、全ＣＰＵに同一アクセス空間を割り当てた時の各ＣＰＵからの重複アクセスを回避することができる。

【００３４】従って、マルチＣＰＵを対象としたランダム命令試験で、各ＣＰＵが同一ブロックをアクセスでき、かつ別のＣＰＵからデータが破壊されないようにしてキャッシュコヒーレンシィ試験を実現できる。

【００３５】(b)：前記(2)では、前記情報処理装置の試験を行う際、同一論理空間を持ち、命令により切り換え可能な異なるコンテキストを複数用意する。そして、コンテキスト毎に、それぞれ論理アドレスと実メモリの物理アドレスとの対応情報を持ち、ランダム命令列を実行させる過程で、特定命令によりコンテキストを切り換えた際、同一論理アドレスをアクセスしても、実メモリの異なる物理アドレスを指すようにしてキャッシュリプレース（例えば、Ｌ１$からＬ２$へのキャッシュデータの追い出し）を行わせる。このようにすれば、キャッシュリプレース回数をより多く生成することができる。

【００３６】(c)：前記(3)では、前記情報処理装置の試験を行う際、各ＣＰＵが実行するランダム命令列内に試験対象キャッシュアドレスの選択を行う特定命令（例えば、Select_Line（））を設けておき、この特定命令の実行により、各C.PUとも、同一の試験対象キャッシュアドレスを選択し、このアドレスに、各ＣＰＵに割り振ったシフトバイトを加算した値を求め、この値により、各ＣＰＵから見て、同一キャッシュアドレスで、かつ、アクセスが重複しないようにする。

【００３７】すなわち、複数ＣＰＵから試験対象となる同一メモリブロック内でアクセスが重複しないメモリアドレスを導出することができる。このようにすれば、ランダム命令列に意図したキャッシュコヒーレンシィ論理を組み込む手法を実現でき、試験時の全組み合わせの実現時間を短縮させることが可能になる。

【００３８】(d)：前記(4)では、前記情報処理装置の試験を行う際、生成したランダム命令列を別の空間に複写する処理と、両方のランダム命令空間に対して、論理アドレスが等しく、物理アドレスが異なるアドレス変換

テーブルを作成し、該テーブルに異なるコンテキスト値を定義する処理と、初期値として、コンテキストに値を設定し、或る論理アドレスから試験を開始させる処理と、或る割り込みを契機に、命令の論理アドレスを前記アドレス変換テーブルから選択し、この値にランダム命令内アドレスを命令カウンタに設定する処理と、或る割り込みを契機に、コンテキストの値を切り換えることにより、論理アドレスの命令カウンタは変わらずに、物理アドレスのみ変更できるようにする処理とを行う。

【００３９】そして、前記処理により、ランダム命令列を頭から順次実行し、かつ実行空間のみ変化させることで、命令キャッシュのキャッシュコヒーレンシィ試験を可能にする。

【００４０】(e) ：前記(5) では、記録媒体のプログラムを読み出して実行することにより、ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各ＣＰＵ毎に一定値づつずらした値に設定し、前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておく。

【００４１】このようにすれば、全てのＣＰＵから同一命令を実行した時に、アクセス命令のアクセスアドレスは、ＣＰＵ毎に必ず決められたアドレス範囲を指すことができる。すなわち、全ＣＰＵに同一アクセス空間を割り当てた時の各ＣＰＵからの重複アクセスを回避することができる。

【００４２】従って、マルチＣＰＵを対象としたランダム命令試験で、各ＣＰＵが同一ブロックをアクセスでき、かつ別のＣＰＵからデータが破壊されないようにしてキャッシュコヒーレンシィ試験を実現できる。

【００４３】

【発明の実施の形態】以下、発明の実施の形態を図面に基づいて詳細に説明する。

【００４４】§１：システムの説明・・・図２参照

図２はシステム構成図である。このシステムは、マルチＣＰＵシステム（マルチＣＰＵ構成の情報処理装置）の１例であり、バス上に複数（ｎ個）のＣＰＵ（ＣＰＵ－０、ＣＰＵ－１・・・ＣＰＵ－ｎ）と主記憶メモリ１（全ＣＰＵで共有する１個のメモリ）と、システムコントローラ２等が搭載されている。そして、各ＣＰＵは、それぞれキャッシュメモリ（以下、単に「キャッシュ」とも記す）を持っている。

【００４５】この場合、各ＣＰＵ内部（ＣＰＵチップの内部）にはＬ１＄（一次キャッシュ）を設け、各ＣＰＵの外側（ＣＰＵチップとは別のチップ）にはＬ２＄（二次キャッシュ）を設け、これらをバスにより接続している。なお、この例の場合、一次キャッシュと二次キャッシュを設けた例であるが、一般的には更に多くのキャッシュ（三次キャッシュ、四次キャッシュ等、一般的にはｎ次キャッシュまで）設けることもできる。

【００４６】前記Ｌ１＄は小容量のキャッシュであり、例えば、３２ＫＢ（キロバイト）のキャッシュメモリ（それぞれ、Ｗａｙ－０、Ｗａｙ－１、Ｗａｙ－２、Ｗａｙ－３とする）を４個で構成する（４つのキャッシュ領域で構成する）。また、前記Ｌ２＄は１ＭＢ、又は２ＭＢ、又は４ＭＢ等の容量を持つ大容量（Ｌ１＄に比べて大容量）のキャッシュである。この場合、図面上では、Ｌ１＄のＷａｙ－０、Ｗａｙ－１、Ｗａｙ－２、Ｗａｙ－３の各左端がアドレス＝０で、右端がアドレス＝３２Ｋとする。そして、図２の斜線部分は、６４バイト幅のキャッシュラインである。

【００４７】前記バスには、システムコントローラ２が接続され、該システムコントローラ２に接続された別のバスには、入出力制御部（以下「Ｉ／Ｏ制御部」と記す）５を介してＲＯＭ６等が接続されている。前記システムコントローラ２は、システム内のキャッシュ管理等を含む各種制御を行うものであり、該システムコントローラ２内に、キャッシュ情報テーブル４を備えたキャッシュ制御部３が設けてある。

【００４８】前記キャッシュ制御部３は、各ＣＰＵが持つＬ１＄、Ｌ２＄の情報（キャッシュ情報）をキャッシュ情報テーブル４に格納し、これらＬ１＄、Ｌ２＄のキャッシュ管理（又はキャッシュ制御）を行う機能を備えている。例えば、ＣＰＵが主記憶メモリ１のデータを取り込む（ロードする）場合、先ず、内部のＬ１＄を見にゆく。

【００４９】そして、Ｌ１＄、Ｌ２＄に該当するデータがなければ、他のＣＰＵへ該当するデータを取りに行くが、この時、前記ＣＰＵはシステムコントローラ２へキャッシュ情報を貰いに行き、その情報に基づいて他のＣＰＵからキャッシュデータを読み込む。

【００５０】従って、システムコントローラ２は、各ＣＰＵのＬ２＄の情報を、キャッシュ情報テーブル４に格納し、前記情報を常に更新しながら管理しておき、各ＣＰＵに対してこの情報によりキャッシュ制御を行うようになっている。また、ＲＯＭ６には、以下に説明するキャッシュ試験用のプログラムが格納されている。

【００５１】そして、以下に説明するキャッシュコヒーレンシィ試験を行う場合には、ＲＯＭ６に格納されているプログラムを、ＩＰＬでＣＰＵ（例えば、ＣＰＵ－０）が主記憶メモリ１へロードした後、該ＣＰＵが主記憶メモリ１のプログラムを取り込んで実行することで、このシステム内での試験を行う。

【００５２】前記システムにおいて、各ＣＰＵがロード命令（Load）、及びストア命令（Store ）を実行する場合の処理概要は次の通りである。例えば、前記ロード命令は、主記憶メモリ１のデータをＣＰＵ内のレジスタに取り込む処理であるが、この場合、最初の（Ｌ１＄、Ｌ２＄が全て空の状態での）ロード命令であれば、データは、主記憶メモリ１→Ｌ２＄→Ｌ１＄→ＣＰＵ内部のレ

ジスタの順に転送される。

【００５３】この場合、例えば、主記憶メモリ1の１００番地のデータをロードするのであれば、主記憶メモリ1の１００番地から６４バイト分のデータを読み出し、このデータをＬ２＄に格納し、その後、該Ｌ２＄からＬ１＄へデータを転送し、Ｌ１＄のＷａｙ－３に格納する。

【００５４】このように動作を繰り返すことで、順次データのロードを行うと、Ｌ１＄には、順次Ｗａｙ－３→Ｗａｙ－２→Ｗａｙ－１→Ｗａｙ－０にそれぞれ６４バイト分のデータが格納され、図2に示したキャッシュラインに、主記憶メモリ1から読み出したデータが順次格納されるようになる。

【００５５】また、ストア命令は、ＣＰＵ内のレジスタに格納されているデータを主記憶メモリ1に書き込む処理であるが、この命令を実行した場合の処理は次の通りである。先ず、ＣＰＵ内のレジスタに格納されているストアすべきデータに対応するデータがＬ１＄に有るかどうかを判断し、該データが有れば、Ｌ１＄の対応するデータを新データ（前記ストア対象のデータ）で書き換えて処理を終了する。この時、主記憶メモリ1のデータは書き換えないため古いデータのままである。

【００５６】しかし、Ｌ１＄に前記ストア対象のデータに対応するデータが無ければ、ストアすべきデータに対応するデータがＬ２＄に有るかどうかを判断し、該データが有れば、Ｌ２＄の対応するデータを新データ（前記ストア対象のデータ）で書き換えて処理を終了する。この時、主記憶メモリ1のデータは書き換えないため古いデータのままである。

【００５７】更に、前記ストアすべきデータに対応するデータが、Ｌ１＄にもＬ２＄にも無ければ、前記対応するデータを主記憶メモリ1から読み出してＬ２＄に書き込み、Ｌ２＄の対応するデータを新データ（前記ストア対象のデータ）で書き換えて処理を終了する。この時、主記憶メモリ1のデータは書き換えないため古いデータのままである。以下、キャッシュコヒーレンシィ試験の具体例を詳細に説明する。

【００５８】§２：例1の説明・・・図3、図4参照

図3は例1の説明図（その1）であり、A図はオペランドアドレス生成用ベースレジスタの定義、B図はオペランドアドレス生成用インデックスレジスタの定義、C図はオペランドアドレス生成用Displacement値の定義を示す。また、図4は例1の説明図（その2）である。

【００５９】例1は、全ＣＰＵに同一アクセス空間を割り当てた時の各ＣＰＵからの重複アクセスを防ぐ手法を用いた試験方法である。なお、各ＣＰＵには、それぞ各種のレジスタ等が設けてあるが、以下の説明では、各ＣＰＵ内に設けたインデックスレジスタを％Ｇ１、ベースレジスタを％Ｇ２と記す（％Ｇ１、％Ｇ２はいずれもレジスタの番号）。

【００６０】主記憶メモリ1に対するアクセス命令のオペランドアドレスの生成は、①：ベースレジスタ（％Ｇ２）のアドレス+インデックスレジスタ（％Ｇ１）のアドレス（％Ｇ２+％Ｇ１）、又は、②：ベースレジスタ（％Ｇ２）のアドレス＋ディスプレースメント値（％Ｇ２＋ディスプレースメント値）で決まる。従って、％Ｇ１の値、％Ｇ２の値、及びディスプレースメント値を規定することで、各ＣＰＵのアクセスする空間（主記憶メモリ1の領域）を同じにして、かつ領域の重複アクセスを回避させることができる。そこで、各レジスタの定義を次のようにする。

【００６１】(1)：オペランドアドレス生成用ベースレジスタの定義

ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタ（％Ｇ２）の値を、各ＣＰＵ毎に固定に定義し、各ＣＰＵはそのレジスタ（％Ｇ２）の値に１６バイトづつずらした値を設定する。なお、ここで定義した１６バイトという値は、キャッシュのアクセス単位が６４バイトの時に最大４ＣＰＵを動作させた場合の値である。つまり、試験対象のＣＰＵ数とキャッシュのアクセス単位によりこの値は変化する。

【００６２】この例では、ＣＰＵ－０～ＣＰＵ－３の各ベースレジスタ（％Ｇ２）の値を図3のA図のように設定する（図のｘは１６進数のヘキサを意味する記号である）。例えば、ＣＰＵ－０内のベースレジスタ（％Ｇ２）のアドレスをｎ＝「０ｘ００１００００」に設定し、この値に１６バイトづつずらした値を他のＣＰＵのベースレジスタに設定する。

【００６３】すなわち、ＣＰＵ－１内のベースレジスタ（％Ｇ２）のアドレス＝ｎ＋１６＝「０ｘ００１０００１０」に設定し、ＣＰＵ－２内のベースレジスタ（％Ｇ２）のアドレス＝ｎ＋３２＝「０ｘ００１０００２０」に設定し、ＣＰＵ－３内のベースレジスタ（％Ｇ２）のアドレス＝ｎ＋４８＝「０ｘ００１０００３０」に設定する。

【００６４】(2)：オペランドアドレス生成用インデックスレジスタの定義

ランダム命令列で生成されるアクセス命令のオペランドアドレスのインデックス部を指すインデックスレジスタ（％Ｇ１）の値を固定に定義し、その内容をキャッシュのアクセス境界（６４バイトバウンダリ）に設定する。この例では、図3のB図に示したように、インデックスレジスタ（％Ｇ１）に、「０ｂｍ・・・ｍｍｍ０００００」を設定する。この場合、例えば、０ｘ３ｃ０（ｍの単位はビット）とする。

【００６５】(3)：オペランドアドレス生成用ディスプレースメント値（直値）の定義

ランダム命令列で生成されるアクセス命令のオペランドアドレス内のディスプレースメント値のビット４、５を０固定とする。この例では、ディスプレースメント値は

「0bnnnnnnnn00mmmm」に設定する。この場合、例えば、ディスプレースメント値=0x2c8（m，nの単位はビット）とする。また、前記値「0bnnnnnnnn00mmmm」において、図示の右端のmはビット0、その左側のmはビット1となる。従って、ビット4は右端から5ビット目の0である。

【0066】なお、前記ビット4、5は、前記(1) のオペランドアドレス生成用ベースレジスタの定義で説明した条件（前記定義した16バイトという値は、キャッシュのアクセス単位が64バイトの時に最大4CPUを動作させた場合の値である。つまり、試験対象のCPU数とキャッシュのアクセス単位によりこの値は変化する。）により決定する。

【0067】この場のmの値は、生成された命令の種類により規定される。例えば、1バイト格納命令が生成された場合は、「mmmm」の値（mは任意の値）、2バイト格納命令が生成された場合は、「mmm0」の値、4バイト格納命令が生成された場合は、「mm00」の値、8バイト格納命令が生成された場合は、「mm0」の値とする。

【0068】すなわち、そのアドレスからデータをアクセスした時に、16のバウンダリを越えない値を定義する必要がある。16バウンダリを越えてアクセスすると、次のCPUがアクセスする領域にアクセスが及んでしまうからである。前記の条件で生成されたアクセス命令のオペランドアドレスは、必ず64バイトバウンダリでCPU単位に決められたアドレスを指すことができる。例えば、以下の命令が生成された場合で前記の例の値を当てはめると、図3のC図に示したようになる。

【0069】①：ベースレジスタ（%G2）のアドレス+インデックスレジスタ（%G1）のアドレスを適用した場合、次のようになる。例えば、8バイトアクセスの例では、ストア命令をSTX（X：例えば、8バイト）、CPU内に設けられたIレジスタ番号5を%I5、アクセス命令のアクセスアドレス（オペランドアドレス）をOPとすると、STX，%I5，［%G2+%G1］は、次のようになる。なお、前記［%G2+%G1］は主記憶メモリ1のアドレスを示す。

【0070】すなわち、CPU-0が、%G2=0x00100000+%G1=0x3c0、OP=0x001003c0、CPU-1が、%G2=0x00100010+%G1=0x3c0、OP=0x001003d0、CPU-2が、%G2=0x00100020+%G1=0x3c0、OP=0x001003e0、CPU-3が、%G2=0x00100030+%G1=0x3c0、OP=0x001003f0となる。

【0071】②：ベースレジスタ（%G2）のアドレス+ディスプレースメント値のアドレスを適用した場合、次のようになる。例えば、8バイトアクセスの例では、ストア命令をSTX（X：例えば、8バイト）、Iレジスタ番号5を%I5、オペランドアドレスをOPとすると、STX，%I5，［%G2+0x2c8］は、次のようになる。なお、%I5は、オペランドアドレスを格納するレジスタである。

【0072】すなわち、CPU-0が、%G2=0x00100000+Imm13=0x2c8、OP=0x001002c8、CPU-1が、%G2=0x00100010+Imm13=0x2c8、OP=0x001002d8、CPU-2が、%G2=0x00100020+Imm13=0x2c8、OP=0x001002e8、CPU-3が、%G2=0x00100030+Imm13=0x2c8、OP=0x001002f8となる。なお、前記［%G2+0x2c8］は主記憶メモリ1のアドレスを示す。

【0073】前記のように、全てのCPUから同一命令を実行した時に、前記OPで表されるアクセス命令のアクセスアドレスは、CPU毎に必ず決められた範囲（この場合は、16バイト）を指すことができる。従って、各CPUでランダムに発行したアクセス命令（例えば、ストア命令）は、全CPUに同一アクセス空間（メモリ領域）を割り当てた時に、各CPUからの重複アクセスを防止できることになる。この状態を図4に示す。

【0074】図4では、CPU-0～CPU-3は、それぞれ16バイトずつ離れた領域（アドレス範囲）に割り当てられており、アクセスする場合の重複が防止できている。すなわち、全CPUに同一アクセス空間（メモリ領域）を割り当てた時に、各CPUからの重複アクセスを確実に防止することが可能になる。

【0075】この場合、主記憶メモリ1へアクセスする際のデータのブロック単位は、例えば64バイトであるが、一般的には、（ブロックサイズ）／（CPU数）＝各CPUに割り当てられたアドレス範囲（図4では16バイト）である。

【0076】§3：例2の説明・・・図5参照

図5は例2の説明図であり、アクセス用レジスタ番号とメモリアドレスの関係を示している。例2は、ウインドウの切り換えだけでキャッシュコヒーレンシィの動作環境を実現することで、ランダム命令列にキャッシュコヒーレンシィの動作環境をより多く生成させる例である。

【0077】例えば、「Ultra-sparc」等のRISC（Reduced Instruction Set Computer）型コンピュータ（縮小命令セット・コンピュータ）には、汎用レジスタ（%G0～%G7）と各ウインドウ毎にインレジスタ（%I0～%I7）、ローカルレジスタ（%L0から%L7）と、アウトレジスタ（%O0～%O7）があり、ウインドウレジスタは、図5に示したように、隣り合うウインドウのイン（%I）とアウト（%O）のレジスタが重複している（%Iと%Oは内容が同じ）。

【0078】この場合、ローカルレジスタ（%L0～%L7）はウインドウ（CW0～CW4）を切り換えると

異なる内容となるが、隣り合うウインドウのアウトレジスタ（%O）とインレジスタ（%I）の内容は同じである。これらのレジスタは、プログラムが或るウインドウを指している時は、その時のウインドウのレジスタと汎用レジスタ（%G）のみが使用可能となる。

【0079】そのため、アクセス用として定義するのは、どのウインドウからでも参照できる汎用レジスタ（%G）を使用しているケースが殆どである。この場合、多くても7つのアドレスしか指定できない。この発明では、アクセスレジスタとして着目したのは、各ウインドウのローカルレジスタ（%L0～%L7）であり、ウインドウが5つあると仮定すると、40個のアドレスが指定可能となる（ウインドウの切り換えだけで、極めて多くのレジスタ値が簡単に得られる）。

【0080】この40個のレジスタに、予め、論理空間の異なるアドレスを設定しておくことにより、命令でカレントウインドウを切り換えるだけで、キャッシュコヒーレンシィの動作環境がより多く作れる。この場合、カレントウインドウCW0～CW4（CW：Current Window）は、プログラム上のウインドウ切り換え命令で切り換え可能なウインドウである。そして、%Lのレジスタを CWで切り換えることにより、ベースレジスタとして使用する。すなわち、%LをCW0～CW4で切り換えることにより、該%Lを前記ベースレジスタとして使用する。

【0081】また、前記%Gは前記インデックスレジスタとして使用する。この場合、%Gには、インデックスポインタとして、－64、0、＋64の値が格納されているので、これを使用すれば、CWの切り換えだけでキャッシュコヒーレンシィの動作環境を簡単に実現することが可能になる。

【0082】このように、各ウインドウ毎に設けられたローカルレジスタ（%L）に、予め、論理空間の異なるアドレスを設定しておき、このローカルレジスタをウインドウにより切り換えて、前記ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタとして使用し、前記汎用レジスタ（%G）を前記オペランドアドレスのインデックス部を指すインデックスレジスタとして使用することにより、ウインドウの切り換えだけで、キャッシュコヒーレンシィの動作環境を実現できるようにした。

【0083】なお、この場合には、主記憶メモリ1に対するアクセス命令のオペランドアドレスの生成は、①：ベースレジスタ（%L）のアドレス+インデックスレジスタ（%G）のアドレス（%L+%G）で決まる。

【0084】§4：例3の説明・・・図6参照

図6は例3の説明図であり、L1$リプレースメント用レジスタの定義（CW0，1，2，3，4）を示している。図6は図5に示したローカルレジスタ%Lを拡大した説明図である。例3は、キャッシュリプレース（キャッシュデータの置き換え）の回数をより多くすることにより、ランダム命令列にキャッシュコヒーレンシィの動作環境をより多く生成させる例である。

【0085】ところで、アクセスするアドレスレジスタの数を多く定義しても、実際にアクセスするアドレスがアクセスレジスタ+（－）0x10000のように範囲が大きすぎると、キャッシュの該当ライン（キャッシュライン）にアクセスする確率が非常に少なくなり、結果としてキャッシュリプレース（キャッシュデータの追い出し）の機会が少なくなってしまう。

【0086】この場合のキャッシュリプレースは、キャッシュデータの置き換え、すなわち、L1$からL2$へのデータの追い出しを意味する。例えば、L1$に新たなデータを書き込む場合、L1$の領域に空きがなければ、L1$内の古いデータから追い出してL2$へ書き込み、この時追い出しにより空いたL1$内の領域に新たなデータを書き込む。

【0087】そこで、アドレスを決定する条件である、インデックスレジスタ（%L）と、ディスプレースメント値（イミーディエイト値）を、マイナス64、0、プラス64（－64，0，＋64）の3箇所に固定し、それ以外の値が生成されないようにする。これにより、アクセス域は、ローカルレジスタ（%L）の指すアドレスのプラス、マイナス64バイト以内に限定される。

【0088】この場合、ベースレジスタとして使用する%L0～%L7（図5のレジスタ%Lを図6に拡大して示してある。）は、それぞれL1$のWay－0～Way－3のそれぞれの幅である32KBに合わせ、32KBずつ離して設ける。例えば、%L0と%L1の間隔は32KB、%L1と%L2の間隔は32KB、%L2と%L3の間隔は32KBとなっている。

【0089】そして、各%L毎に、インデックスレジスタとして使用する汎用レジスタ%Gは、32KBのバウンダリから、－64バイト、0バイト、＋64バイトの間隔に設定されている。従って、インデックスレジスタとして使用する汎用レジスタ%Gにはインデックス値を入れるが、この場合、%G1にインデックス値＝－64を入れ、%G2にインデックス値＝＋0を入れ、%G3にインデックス値＝＋64を入れる。

【0090】また、ディスプレースメント値（イミーディエイト値）も、－64バイト、0バイト、＋64バイトの値を使用する。このようにして、ベースレジスタ（%L）の値に、インデックスレジスタ（%G）の値、或いはディスプレースメント値（イミーディエイト値）を加算して、アクセス命令のオペランドアドレスを生成することにより、アクセス命令を生成する。

【0091】従って、各CPUで無作為に生成された命令列で、4回アクセス命令を実行すれば、確率的に1回は同一キャッシュラインを指すことになる。この場合のキャッシュラインは、図6に示したL1$内のWay－

0～Way-3に跨がって斜線で図示したラインである。

【0092】§5：例4の説明・・・図7参照

図7は例4の説明図であり、試験空間とMMUテーブルとの関係を示している。例4は、キャッシュリプレースの回数をより多く生成することにより、ランダム命令列にキャッシュコヒーレンシィの動作環境をより多く生成させる例である。

【0093】キャッシュリプレース回数をより多くする方法として、コンテキストの切り換えがある。なお、例4において、コンテキストとは連続した1つの論理空間の固まりのことを言う（UNIXマシンの用語）。また、MMUはアドレス変換テーブルのことである。

【0094】図7に示したように、同一論理空間（論理アドレス＝0～n）を持ち、異なるコンテキストCtx（この場合、Ctx=a、Ctx=b、Ctx=c）をN個（この例ではN＝3）用意する。この場合、各コンテキスト毎に、論理アドレス0～nまでの論理空間を有し、それぞれ、論理アドレスと物理アドレスとの対応表を持っている。そして、それぞれの論理アドレスと物理アドレスとの対応表には異なるデータを設定しておく。

【0095】例えば、図7のCtx=a、Ctx=b、Ctx=cの論理空間には、論理アドレスと物理アドレスが対応づけられているので、この対応アドレス情報を用いれば、コンテキストの切り換えにより、図示矢印で示すように、論理アドレスが同じでも、異なる物理アドレス（実メモリのアドレス）を指す。

【0096】従って、前記環境でランダム命令列を実行させ（最初のコンテキストは、Ctx=aとする）、或るところでコンテキストを切り換える（例えば、Ctx=aからCtx=bに切り換える）ことにより、コンテキストを変える前後では論理アドレスは同じであるが、コンテキストが異なることから、同一論理アドレスをアクセスしても、キャッシュリプレースは行われる。

【0097】前記のように、キャッシュリプレースさせる環境を組み合わせることにより、従来のキャッシュコヒーレンシィの発生回数に比べ、桁違いに多くのキャッシュコヒーレンシィ状態を実現させることが可能になる。従って、短期間でキャッシュコヒーレンシィの検証が可能となり、ハードウェアの品質も保証できるようになる。

【0098】§6：例5の説明・・・図8参照

図8は例5の説明図であり、A図は試験命令の説明図、B図はキャッシュアドレステーブルを示す。例5は、ランダム命令列に意図したキャッシュコヒーレンシィ論理を組み込むことにより、複数CPUから試験対象となる同一キャッシュラインを導く例である。

【0099】ところで、前記例1～例4までの試験方法では、全CPUからランダムにアクセス命令を発行することにより、各CPU内のキャッシュと主記憶メモリ1間の全状態の組み合わせを実現させることはできるが、この方法では、どれだけ流せば全組み合わせが実現できるか定かでない。そこで、この全状態の組み合わせの実現時間を短縮させるための手法として、次のような例5の手法がある。

【0100】試験対象キャッシュアドレスの選択は、セレクトライン命令「Select_Line()」で与えられる。この試験では、試験対象となるキャッシュアドレスを選択し、各CPUに割り振ったシフトバイトを加算した値を求める。この場合、全CPUが同一物理アドレスを指すようにする必要があるため、次のような方法を用いる。

【0101】①：各CPU毎に、試験キャッシュアドレスのテーブルを作成する。このテーブル内の論理アドレスは、CPU間で異なっても良いが、物理アドレスは必ず一致させる必要がある。

【0102】②：各CPUから呼び出されたセレクトライン命令（関数）「Select_Line()」は、呼び出し元のリンクアドレス（A）の下位の数ビットをインデックスとし、キャッシュアドレステーブル（B）から対象アドレス（論理アドレス）を求める。この場合、各CPUは同一アドレス（A）からセレクトライン命令（関数）「Select_Line()」を呼ぶため、各CPUが選択したアドレスは全て同一物理アドレスを指す。

【0103】③：選ばれた論理アドレスから1ブロック（64バイト）を、各CPUで分割するため、自CPU番号に16を掛けた値を足し込む。このようにして選択された試験アドレスは、各CPUから見て同一メモリブロック（64バイト）内で、かつ、アクセスが重複しない値となる。

【0104】この場合、前記セレクトライン命令（関数）は、Select_Line()＝{C=*(B+(A&0xf8)；D=C+(CPU番号×16)；Return(D)}で表される。この場合、Aはリンクアドレス、Bは論理アドレス、Cは物理アドレス、Dは16バイトずらしたアドレス、16はずらすバイト数、0xf8は下位のビット、*は括弧内の内容、Return(D)はDの内容を持ってリターンすることを意味している。

【0105】すなわち、Select_Line()＝{C=*(B+(A&0xf8)；D=C+(CPU番号×16)；Return(D)}は、Bの論理アドレス+（アドレスA+下位のビット「0xf8」）の内容をもってCの値とし、このCの値に（CPU番号×16）の値を足した値をDとし、このDの値を持ってリターンする、という内容である。

【0106】また、図8のA図では、各CPUが同一命令（テスト命令、或いは試験命令）を実行することを示している。そして、図8のB図では、各CPU毎の論理アドレスBが異なっていても、その論理アドレスから変換される物理アドレス（キャッシュの物理アドレス）は

全て同じアドレスになることを示している。

【0107】例えば、CPU－0の論理アドレス「500000」は物理アドレス「200000」に変換され、CPU－1の論理アドレス「7000000」は物理アドレス「200000」に変換される。また、CPU－0の論理アドレス「5002000」は物理アドレス「202000」に変換され、CPU－1の論理アドレス「7002000」は物理アドレス「202000」に変換される。

【0108】このように、各CPUが実行する前記ランダム命令列内に試験対象キャッシュアドレスの選択を行う特定命令（セレクトライン命令）を設けておき、この特定命令の実行により、各CPUとも、同一の試験対象キャッシュアドレスを選択し、このアドレスに、各CPUに割り振ったシフトバイトを加算した値を求め、この値により、各CPUから見て、同一キャッシュアドレスで、かつ、アクセスが重複しないようにできる。すなわち、前記セレクト関数により選択されたアドレスは、各CPUから見て、同一メモリブロック内で、アクセスが重複しないようにすることができる。

【0109】§7：例6の説明・・・図9参照

図9は例6の説明図であり、（a）はルーチンテーブル例、（x）はマスタCPU処理（CPU－0）、（y）はスレーブCPU処理（CPU－1，2，3）を示す。この場合、CPU－0をマスタとし、他のCPUをスレーブとしているが、マスタCPUはいずれのCPUでも良い。例6は、ランダム命令列に意図したキャッシュコヒーレンシィ論理を組み込むことにより、意図したキャッシュコヒーレンシィ論理の検証がランダム命令試験で実現可能にした例である。

【0110】図9に示したようなキャッシュコヒーレンシィ（キャッシュの一致性）を試験するルーチンを予め用意する。例えば、（a）に示したようなルーチンテーブルを用意する。この（a）に示したルーチンでは、IF文を使い、各CPU専用処理部（x）、（y）を設ける。これは全CPUからこのルーチンに起動がかかるためであり、かつ、CPU毎に異なる動作をさせるためである。

【0111】このCPU毎の専用処理部（x）、（y）に、キャッシュコヒーレンシィの論理（従来例で説明した「MOESI」として周知）を組み込んだこのルーチンを、ランダム命令列の生成過程で無作為に展開することにより、意図的なキャッシュコヒーレンシィ試験が実現可能になる。以下に、前記ルーチン内の詳細論理を説明する。

【0112】先ず、試験対象キャッシュライン（図2の斜線部分参照）を選択（Select_Line命令で選択）し、CPUの同期をとった後、各CPUの命令列に分岐する（CPU毎に、処理を分けて実行する）。ここでは定義上、CPU－0をマスタ、その他のCPUをスレーブとする。前記マスタはキャッシュに或る状態を設定する。例えば、ストア命令（Store ）を使用する。スレーブは、マスタが設定したキャッシュの状態に対し、変化を与える。例えば、ロード命令（Load）により変化を与える（データを変化させる）。

【0113】この例では、マスタが最新のキャッシュ情報を持っている時に、スレーブからその情報を読み出した時のキャッシュ状態の変化を実現させている。このようにして、前記ストア命令と、ロード命令を変化させることにより、多種多様なキャッシュの状態変化を明確に実現でき、その時のキャッシュコヒーレンシィ（キャッシュの一致性）の検証が可能になる。

【0114】このように、前記（a）に示すルーチンを、ランダム命令列生成時に無作為に挿入することにより意図したキャッシュコヒーレンシィ論理の検証がランダム命令試験で実現可能となる。例えば、前記キャッシュコヒーレンシィは、次のような内容である。すなわち、データは、主記憶メモリ1や各CPUのL1$、L2$に格納されているが、最新のデータは必ずしも主記憶メモリ1にある必要はなく、いずれかのCPUのL1$、或いはL2$に格納されていることもある。

【0115】つまり、最新のデータは前記主記憶メモリ1、或いは各CPUのL1$、或いはL2$のいずれかにあれば良く、どこにあっても良いが、必ずどこかになければならない。これらの論理は全て、「MOESI」として確立された論理に従って、処理されている。

【0116】§8：例7の説明・・・図10、図11参照

図10は例7の説明図（その1）であり、A図は中間言語の命令定義体、B図はテーブル例である。また、図11は例7の説明図（その2）であり、（x）はマスタCPU処理（CPU－0）、（y）はスレーブCPU処理（CPU－1、2、3）を示す。

【0117】例7は、ランダム命令列に意図したキャッシュコヒーレンシィ論理を組み込むことにより、キャッシュコヒーレンシィ試験における、意図したキャッシュデータの組み合わせをランダムに実現させる例である。例7の手法では、意図したキャッシュの動作は1ルーチンで1つ実現できるが、本来の試験はランダム命令試験であることから、意図したキャッシュの動作にランダム性を持たせ、キャッシュ動作の組み合わせが1つのルーチン内で出来ることが望ましい。これを実現する手法を以下に説明する。

【0118】先ず、ルーチン内で定義する図10のA図に示したような中間言語（マクロ命令）の命令定義体を用意する。この中間言語（マクロ命令）はランダム命令列を生成するジェネレータによって解析され、実行形式の命令に変換される。

【0119】この場合、前記中間言語のパラメータの1つとして、次のような機能を持たせる。すなわち、「C

HAIN」,「CHAIN_LIMIT」,「END」の各命令において、「CHAIN」は、以降に命令列が存在することを意味し、「CHAIN_LIMIT」は、以降に命令列が存在することを意味し、かつ、本パラメータで指定した命令列(連続していることが条件)に対して、メモリに展開する際、展開する命令の順番を無作為に入れ換えることを意味する。「END」は、命令列の最終を意味する。

【0120】以上の機能を例を挙げて説明する。図10のB図、及び図11に示した例は、キャッシュコヒーレンシィ試験の1例である。この場合、CPU-0、1、2、3が、それぞれ初期化状態からストア命令(STORE)とロード命令(LOAD)を実行した後に、各CPUからキャッシュ状態を変える幾つかの命令(この例では、Dflush,U _flush,Load,Store,Casxa)を、前記チェインリミット「CHAINLIMIT」を付加して図10のB図に示したテーブルを作成する。

【0121】そして、このテーブルが、前記ジェネレータから選択されると、先頭から順に実行形式の命令(機械語)に置き換えられる。この過程で、図11に示した(b)の命令列を置き換えるが、この時、置き換える命令の順序が変わるため、このテーブルが選択される度に、異なる命令列が生成され、その結果、種々のキャッシュ状態の遷移が実現できる。

【0122】従って、試験の目的である、キャッシュ状態を変化させる命令の種類と、その命令を発行する直前の状態を変えることにより、少ないテーブルで多くのキャッシュ状態遷移試験が実現可能になる。

【0123】§9:例8の説明・・・図12参照

図12は例8の説明図であり、A図はランダム命令の説明図、B図は状態1、C図は状態2である。例8は、命令のキャッシュコヒーレンシィ試験の実現を可能にする手法である。前記例1～例7のようなデータ(命令以外の通常のデータ)のキャッシュコヒーレンシィではなく、命令のキャッシュコヒーレンシィ試験を可能にさせるには、命令実行空間を複数持たせる必要がある。この条件を満足し、かつ、前記例1～例7の機能を有効にする手法は、次の手順により実現する。

【0124】①:生成したランダム命令列を別の空間にそっくり複写する。

【0125】②:両方のランダム命令空間(元の命令空間と複写後の命令空間)に対して、論理アドレスが等しく、物理アドレス(ランダム命令の空間)が異なるMMUテーブル(アドレス変換テーブル)を作成する。そして、各MMUテーブルには、異なるコンテキスト値を定義する。なお、この場合のコンテキストとは、空間を切り換えるための制御レジスタのことであり、コンテキスト値は前記制御レジスタの値である。

【0126】③:初期値として、コンテキスト値=10とし、例えば、論理アドレス=0x000000から試験を開始する。

【0127】④:或る割り込みを契機に、命令の論理アドレスをMMUテーブルから任意に選択する。この値にランダム命令列内のアドレス(ランダム命令列の空間が1MBと仮定した場合は、1MBバウンダリの値をオアしたアドレス)を命令カウンタに設定する。

【0128】例えば、0x5050010番地で割り込みが発生した場合、新命令アドレス(0x7000000)にランダム命令列内アドレス(0x50010)を足した値=0x7050010となる。そして、このアドレスで復帰することにより、ランダム命令列の実行順序が乱されることがなく、かつ論理アドレスを変化させることができる。

【0129】⑤:また、或る割り込みを契機に、コンテキストを切り換えて異なったコンテキスト値(10→20)を得る。これにより、論理アドレスの命令カウンタは変わらずに、物理アドレスのみを変更できる。この場合、0x300000のランダム命令空間の命令が動作する。

【0130】⑥:前記①～⑤の処理により、ランダム命令列は頭から順次実行され、かつ、実行する空間のみ変化することになる。つまり、実行される物理空間、及び実行論理アドレスが順次切り換わるため、命令のキャッシュコヒーレンシィ試験が可能となる。

【0131】なお、L2$が2MBと仮定すると、実アドレスで2MB離れたアドレスをアクセスすると、キャッシュのリプレースが発生する。また、L1$が64KBの4Wayであると仮定すると、16KB離れた論理アドレスを5回アクセスすることでキャッシュのリプレースが発生する。

【0132】§10:試験例例の説明・・・図13、図14参照

図13は試験例の説明図、図14はランダム命令列生成処理フローチャートである。以下、図13、図14に基づいて具体的な試験例を説明する。

【0133】(1) :試験例の説明

キャッシュコヒーレンシィの試験は例えば、次のようにして行う。

【0134】①:全CPUから同一タスクに起動をかける。

【0135】②:起動されたタスクはマスタCPU(この例では、CPU-0)と、スレーブCPU(この例では、CPU-1、2、3・・・n)にCPUを分け、マスタCPUのみ命令の生成を行う。

【0136】③:ランダム命令の生成処理を行う(後述する)。

【0137】④:命令生成後、全CPUで同期をとり、その後、各CPUはレジスタに初期値を設定し、生成されたランダム命令域の先頭に処理を渡す。この場合、初期値を設定するレジスタの中にアクセス命令のアクセス

専用に割り振られたレジスタ（例えば、G1：インデックス専用、G2：ベースレジスタ専用）には、データ域を指す論理アドレスと、そのアドレスからのディスプレートメントを指すためのインデックス値を設定する。また、アクセスの変更は、ランダム命令列から呼び出されたサブルーチンにより変更される。また、何かの割り込みを契機に変更しても良い。

【0138】⑤：ランダム命令列の最後の復帰処理により元のタスクに戻る。

【0139】⑥：最後に試験の結果得られたデータの比較を行うが、このデータ比較の論理は、例えば、次のａ、ｂの通りである。

【0140】ａ：ランダム命令列の実行で或る条件（例えば、割り込み）で全レジスタとメモリの内容をトレースする。この場合、本ランダム命令列を、走行環境を変えて２回実行し、２度目の実行の時、１度目にトレースした情報と比較する。

【0141】ｂ：本タスク処理を実機と、ソフトシュミレータとで実行させ、或る条件で双方のレジスタとメモリの内容を比較する。

【0142】(2) ：ランダム命令の生成処理

以下、図14に基づいてランダム命令の生成処理を説明する。なお、S1～S8は各処理ステップを示す。ランダム命令の生成処理が開始されると、先ず、命令生成域にランダムデータを書き込む（S1）。次に、命令生成域のランダムデータを用い、ルーチンテーブル、又は命令変換テーブルから任意の1つを取り出し、命令に変換後、その命令又は命令列を命令生成域に書き込む。

【0143】この処理を順次行うことで、命令生成域全てにランダム命令を作り上げる。具体的には次の通りである。前記のようにして命令生成域にランダムデータを書き込んだ後、命令生成域の先頭にポインタPを設定する（S2）。そして、前記ポインタPの内容からどちらかに分岐する（S3）。

【0144】一方に分岐した場合は次のような処理を行う。先ず、命令変換テーブルから任意の命令データをランダムに選択し、その中にあるANDと、ORデータと、ポインタPから読み出したランダムデータとを掛け合わせて命令を生成する。生成した命令は、ポインタPのアドレスに格納する。なお、アクセス命令を生成する場合は、決められたアクセス用レジスタが選ばれるように、前記AND／ORデータを定義する（S4）。

【0145】その後、ポインタPの値を更新し（S5）、最終判定を行ない（S6）、最終処理になるまで、前記S3の処理から繰り返す。また、前記S3の処理の結果、他方に分岐した場合は、次のように処理を行う。先ず、ルーチンテーブルから任意のルーチンをランダムに選択し、そのルーチン内のコマンド列を順次命令に変換し、ポインタPのアドレス以降に格納する（S7）。その後、生成した命令の命令数だけポインタPの値を更新し（S8）、前記S6の処理へ移行する。以上の処理を行ない、最終判定で最終処理まで終了したと判定したら、ランダム命令生成処理を終了する。

【0146】§11：試験用プログラムと記録媒体の説明

図2に示した情報処理装置（情報処理システム）は、図2に示した構成の他に、通信制御部、ディスプレイ装置、キーボード、フレキシブルディスクドライブ（フロッピィディスクドライブ）（以下、「FDD」と記す）、CD－ROMドライブ、ハードディスク装置（以下「HDD」と記す）等を備えている。

【0147】そして、図2に示したシステムは、予めROM6に格納（記録、或いは記憶）しておいたプログラムを、CPUの制御により読み出して主記憶メモリ1へ格納し、該CPUが前記プログラムを実行することにより、前記キャッシュコヒーレンシィ試験（情報処理装置の試験）を行う。

【0148】しかし、本願発明は、このような例に限らず、例えば、HDDのハードディスクに、次のようにして試験用のプログラムを格納し、このプログラムをCPUが実行することで前記キャッシュコヒーレンシィ試験を行うことも可能である。

【0149】①：他の装置で作成されたフレキシブルディスク（フロッピィディスク）に格納されているプログラム（他の装置で作成したプログラムデータ）を、FDDにより読み取り、HDDの記録媒体（ハードディスク）に格納する。

【0150】②：光磁気ディスク、或いはCD－ROM等の記憶媒体に格納されているデータを、CD－ROMドライブにより読み取り、HDDの記録媒体（ハードディスク）に格納する。

【0151】③：LAN等の通信回線を介して他の装置から伝送されたプログラム等のデータを、通信制御部を介して受信し、そのデータをHDDの記録媒体（ハードディスク）に格納する。

【0152】

【発明の効果】以上説明したように、本発明によれば次のような効果がある。

【0153】(1) ：キャッシュコヒーレンシィを引き起こすアクセスレジスタの増加、及びキャッシュリプレースの発生確率を増加させた状態で、ランダム命令生成の特質を活かしたキャッシュコヒーレンシィ機能の検証が可能になる。つまり、キャッシュに影響を与えるアクセス命令の前後に、ランダム命令が生成されることで、アクセス命令の発信タイミングをずらすことができ、色々な環境下でキャッシュの状態検証が可能となる。従って、ハードウエアの品質向上が実現できる。

【0154】(2) ：マルチCPUを対象としたランダム命令によるキャッシュコヒーレンシィ試験で、各CPUが同一ブロックをアクセスでき、かつ別のCPUからデ

ータが破壊されないようにしてキャッシュコヒーレンシィ試験を実現できる。

【0155】(3)：マルチCPU構成の情報処理装置を対象とし、該CPU内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各CPU毎に一定値づつずらした値に設定し、前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておく。

【0156】このようにすれば、全てのCPUから同一命令を実行した時に、アクセス命令のアクセスアドレスは、CPU毎に必ず決められたアドレス範囲（例えば、16バイト）を指すことができる。すなわち、全CPUに同一アクセス空間を割り当てた時の各CPUからの重複アクセスを回避することができる。

【0157】従って、マルチCPUを対象としたランダム命令試験で、各CPUが同一ブロックをアクセスでき、かつ別のCPUからデータが破壊されないようにしてキャッシュコヒーレンシィ試験を実現できる。

【0158】(4)：前記情報処理装置の試験を行う際、同一論理空間を持ち、命令により切り換え可能な異なるコンテキストを複数用意する。そして、コンテキスト毎に、それぞれ論理アドレスと実メモリの物理アドレスとの対応情報を持ち、ランダム命令列を実行させる過程で、特定命令によりコンテキストを切り換えた際、同一論理アドレスをアクセスしても、実メモリの異なる物理アドレスを指すようにしてキャッシュリプレースを行わせる。このようにすれば、キャッシュリプレース回数をより多く生成することができる。

【0159】(5)：前記情報処理装置の試験を行う際、各CPUが実行するランダム命令列内に試験対象キャッシュアドレスの選択を行う特定命令を設けておき、この特定命令の実行により、各CPUとも、同一の試験対象キャッシュアドレスを選択し、このアドレスに、各CPUに割り振ったシフトバイトを加算した値を求め、この値により、各CPUから見て、同一キャッシュアドレスで、かつ、アクセスが重複しないようにする。

【0160】すなわち、複数CPUから試験対象となる同一メモリブロック内でアクセスが重複しないメモリアドレスを導出することができる。このようにすれば、ランダム命令列に意図したキャッシュコヒーレンシィ論理を組み込む手法を実現でき、試験時の全組み合わせの実現時間を短縮させることが可能になる。

【0161】(6)：前記情報処理装置の試験を行う際、生成したランダム命令列を別の空間に複写する処理と、両方のランダム命令空間に対して、論理アドレスが等しく、物理アドレスが異なるアドレス変換テーブルを作成し、該テーブルに異なるコンテキスト値を定義する処理と、初期値として、コンテキストに値を設定し、或る論理アドレスから試験を開始させる処理と、或る割り込みを契機に、命令の論理アドレスを前記アドレス変換テーブルから選択し、この値にランダム命令内アドレスを命令カウンタに設定する処理と、或る割り込みを契機に、コンテキストの値を切り換えることにより、論理アドレスの命令カウンタは変わらずに、物理アドレスのみ変更できるようにする処理とを行う。

【0162】そして、前記処理により、ランダム命令列を頭から順次実行し、かつ実行空間のみ変化させることで、命令キャッシュのキャッシュコヒーレンシィ試験を可能にする。

【0163】(7)：前記記録媒体のプログラムを読み出して実行することにより、ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタの値を各CPU毎に一定値づつずらした値に設定し、前記オペランドアドレスのインデックス部を指すインデックスレジスタの値と、前記オペランドアドレス内のディスプレースメント値を予め規定しておく。

【0164】このようにすれば、全てのCPUから同一命令を実行した時に、アクセス命令のアクセスアドレスは、CPU毎に必ず決められたアドレス範囲を指すことができる。すなわち、全CPUに同一アクセス空間を割り当てた時の各CPUからの重複アクセスを回避することができる。

【0165】従って、マルチCPUを対象としたランダム命令試験で、各CPUが同一ブロックをアクセスでき、かつ別のCPUからデータが破壊されないようにしてキャッシュコヒーレンシィ試験を実現できる。

【0166】以上の説明に関して、更に以下の項を開示する。

【0167】(a)：複数のCPUをバスで接続し、各CPUにキャッシュメモリを備えたマルチCPU構成の情報処理装置を対象とし、前記CPU内で、ベースレジスタの値に、インデックスレジスタの値、或いはディスプレースメント値を加算して、アクセス命令のオペランドアドレスを生成することにより、ランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、汎用レジスタと、各ウインドウ毎にインレジスタ、ローカルレジスタ、アウトレジスタが有り、命令が或るウインドウを指している時は、その時のウインドウのレジスタと、汎用レジスタのみが使用可能になる場合、各ウインドウ毎に設けられたローカルレジスタに、予め、論理空間の異なるアドレスを設定しておき、このローカルレジスタをウインドウにより切り換えて、前記ランダム命令列で生成されるアクセス命令のオペランドアドレスを指すベースレジスタとして使用し、前記汎用レジスタを前記オペランドアドレスのインデックス部を指すインデックスレジスタとして使用することにより、命令によるウインドウの切り換えだけで、

キャッシュコヒーレンシィの動作環境をより多く実現する。

【0168】このようにすれば、既存の多数のレジスタを効率的に利用し、ウインドウの切り換えだけでキャッシュコヒーレンシィの動作環境を簡単に実現できる。また、ベースレジスタとして使用できるレジスタの数も多いので、多くの動作環境を簡単に実現できる。

【0169】(b) ：複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内で、ベースレジスタの値に、インデックスレジスタの値、或いはディスプレースメント値を加算して、アクセス命令のオペランドアドレスを生成することにより、ランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、アクセス命令のアドレスを決定する条件である、インデックスレジスタの値と、ディスプレースメント値を、キャッシュメモリの幅で規定される値ｎに対し、－ｎ、０、＋ｎの３か所に固定し、それ以外の値が生成されないようにすることにより、キャッシュリプレースの回数をより多く生成する。

【0170】このように、アクセス命令のアドレスを決定する条件である、インデックスレジスタの値と、ディスプレースメント値を、キャッシュメモリの幅で規定される値ｎ（例えば、ｎ＝６４）に対し、－ｎ、０、＋ｎの３か所に固定し、それ以外の値が生成されないようにすれば、キャッシュリプレースの回数をより多く生成することができる。

【0171】(c) ：複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、キャッシュコヒーレンシィを試験するルーチンを用意し、このルーチンに、各ＣＰＵ毎の専用処理部を設け、該専用処理部にキャッシュコヒーレンシィ試験の論理を組み込んだこのルーチンを、ランダム命令列の生成過程で無作為に展開することにより、意図したキャッシュコヒーレンシィ論理の検証をランダム命令試験で実現可能にする。このようにすれば、意図したキャッシュコヒーレンシィ論理の検証がランダム命令試験で実現可能になる。

【0172】(d) ：複数のＣＰＵをバスで接続し、各ＣＰＵにキャッシュメモリを備えたマルチＣＰＵ構成の情報処理装置を対象とし、前記ＣＰＵ内でランダム命令列を生成してキャッシュコヒーレンシィ試験を行う情報処理装置の試験方法において、マクロ命令の命令定義体を用意し、このマクロ命令をランダム命令列を生成する生成手段によって解析し、先頭から順に実行形式の命令に変換する過程で、展開する命令の順番を無作為に入れ換えることにより、より多くのキャッシュ状態遷移を実現させる。このようにすれば、より多くのキャッシュ状態遷移を実現させ、キャッシュコヒーレンシィ試験を行うことができる。

【図面の簡単な説明】

【図１】本発明の原理説明図である。

【図２】本発明の実施の形態におけるシステム構成図である。

【図３】本発明の実施の形態における例１の説明図（その１）である。

【図４】本発明の実施の形態における例１の説明図（その２）である。

【図５】本発明の実施の形態における例２の説明図である。

【図６】本発明の実施の形態における例３の説明図である。

【図７】本発明の実施の形態における例４の説明図である。

【図８】本発明の実施の形態における例５の説明図である。

【図９】本発明の実施の形態における例６の説明図である。

【図１０】本発明の実施の形態における例７の説明図（その１）である。

【図１１】本発明の実施の形態における例７の説明図（その２）である。

【図１２】本発明の実施の形態における例８の説明図である。

【図１３】本発明の実施の形態における試験例の説明図である。

【図１４】本発明の実施の形態におけるランダム命令列生成処理フローチャートである。

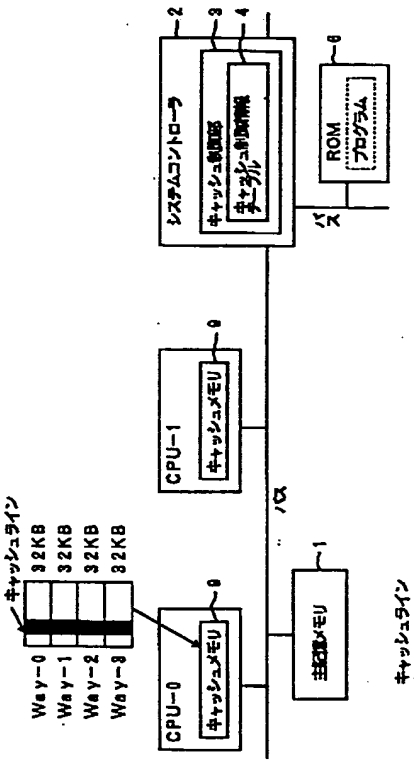
【図１５】従来例のシステム構成図である。

【図１６】従来のランダム命令試験例である。

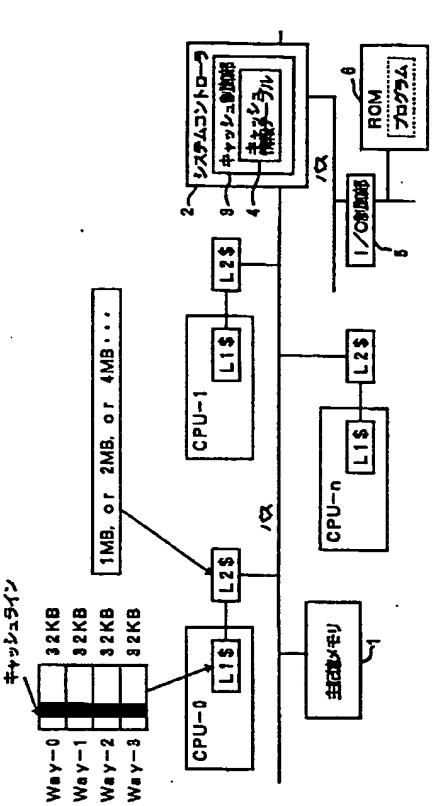
【符号の説明】

１　主記憶メモリ（主記憶装置）
２　システムコントローラ
３　キャッシュ制御部
４　キャッシュ情報テーブル
５　Ｉ／Ｏ制御部（入出力制御部）
６　ＲＯＭ
９　キャッシュメモリ
Ｌ１$　一次キャッシュ（一次キャッシュメモリ）
Ｌ２$　二次キャッシュ（二次キャッシュメモリ）

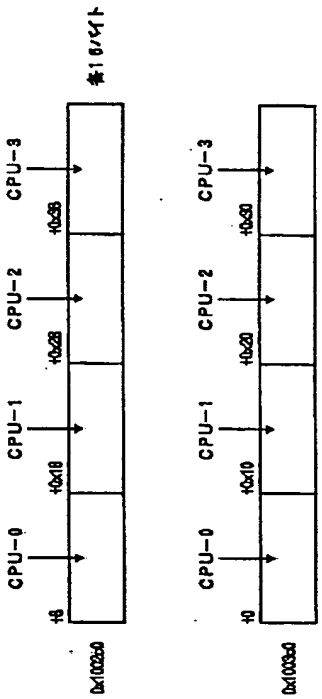
【図1】



【図2】



【図4】



【図9】

例8の説明図

(a) : ルーチンテーブル例

```
Select_line ( ),        / * 対象cache lineの選択 * /
if(CPU_no=0)            / * CPU-0 なら* /
  { (x)   }             / * cache test * /
elseif(CPU_no=1)        / * CPU-1 なら* /
  { (y)   }             / * cache test * /
elseif(CPU_no=2)        / * CPU-2 なら* /
  { (y)   }             / * cache test * /
elseif(CPU_no=3)        / * CPU-2 なら* /
  { (y)   }             / * cache test * /
```

(x) : マスタCPU処理 (CPU-0)

```
(b) Store
```

(y) : スレーブCPU処理 (CPU-1,2,3)

```
(a) Load
```

【図11】

例7の説明図 (その2)

(x) : マスタCPU処理 (CPU-0)

```
(a) Store               (chain)
(b) Load                (chain_Limit)
    Store               (chain_Limit)
    Cassa               (chain_Limit)
    d_flush             (chain_Limit)
    u_flush             (chain_Limit)
```

(y) : スレーブCPU処理 (CPU-1, 2, 3)

```
(a) Load                (chain)
(b) Load                (chain_Limit)
    Store               (chain_Limit)
    Cassa               (chain_Limit)
    d_flush             (chain_Limit)
    u_flush             (chain_Limit)
```

【図3】

例1の説明図 (その1)

A：オペランドアドレス生成用ベースレジスタの定義

ベースレジスタ (%G2)

(CPU-0)
Address = n
0x00100080

(CPU-1)
Address = n+16
0x00100010

(CPU-2)
Address = n+32
0x00100020

(CPU-3)
Address = n+48
0x00100030

B：オペランドアドレス生成用インデックスレジスタの定義

インデックスレジスタ (%G1)

0bm...mmm000000　例：0x3c0 (mの単位はビット)

C：オペランドアドレス生成用Displacement値の定義

0bnnnnnnnnn00mmmm　例：0x2c8 (m, nの単位はビット)

[ベース + インデックス]

STX %I5, [%G2+%G1]
(8バイトアクセス)

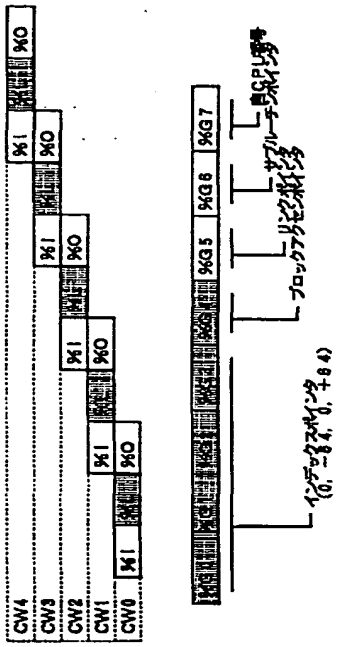
CPU-0 : %G2=0x00100000+ %G1=0x3c0 OP=0x001003c0
CPU-1 : %G2=0x00100010+ %G1=0x3c0 OP=0x001003d0
CPU-2 : %G2=0x00100020+ %G1=0x3c0 OP=0x001003e0
CPU-3 : %G2=0x00100030+ %G1=0x3c0 OP=0x001003f0

[ベース + ディスプレースメント]
STX %I5, [%G2+0x2c8]

CPU-0 : %G2=0x00100000+ Imm13=0x2c8 OP=0x001002c8
CPU-1 : %G2=0x00100010+ Imm13=0x2c8 OP=0x001002d8
CPU-2 : %G2=0x00100020+ Imm13=0x2c8 OP=0x001002e8
CPU-3 : %G2=0x00100030+ Imm13=0x2c8 OP=0x001002f8

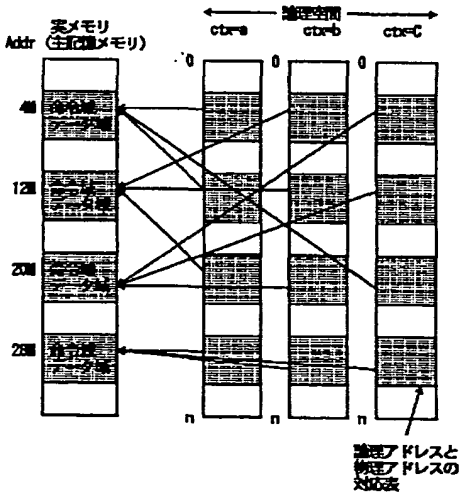
【図5】

例2の説明図



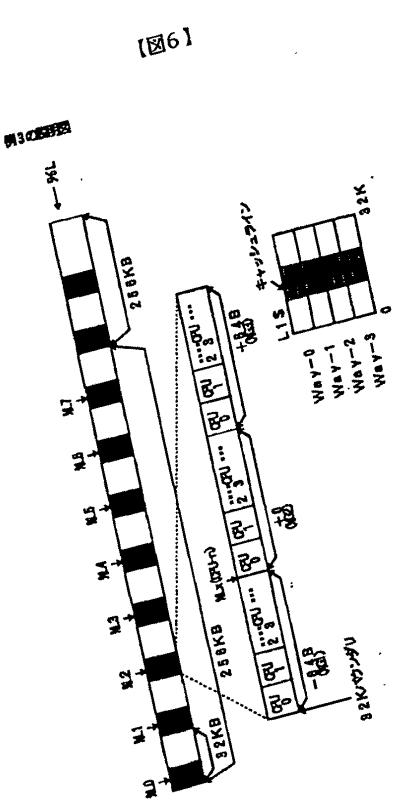
【図7】

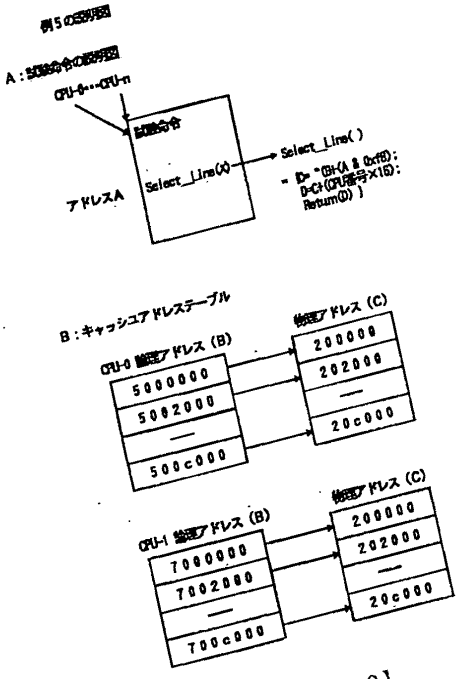
例4の説明図



【図8】

【図6】





【図10】

例7の説明図（その1）

A：中間言語の命令定義体

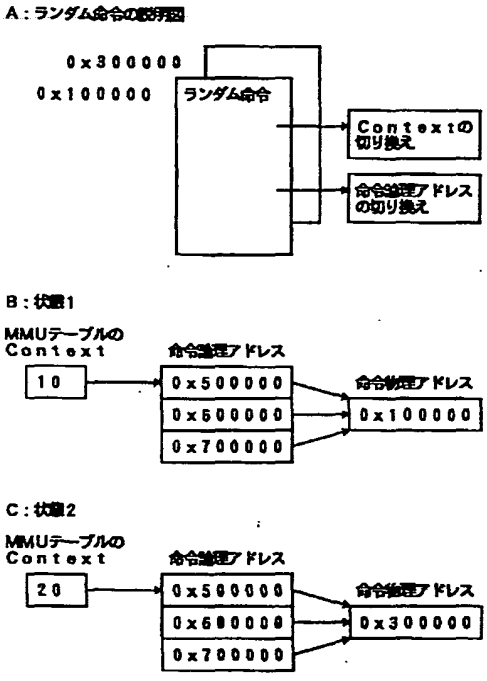
(LOAD(_O5,0,_G0,_L7),1,CHAIN)
(LOAD(_O5,0,_G0,_L7),1,CHAIN_LIMIT)
(STORE(_O5,0,_G0,_L7),1,CHAIN_LIMIT)
(CASXA(_O1,0,0x80,_I4,_I1),1,CHAIN_LIMIT)
(D_FLUSH(_O5,0,_G0,_L7),1,CHAIN_LIMIT)
(U_FLUSH(_O5,0,_G0,_L7),1,CHAIN_LIMIT)
|
(全命令分用意する)

B：テーブル例

```
Select_line0;                   (chain)     /* 対象 cache line の選択 */
if(cpu_num=0)                   (chain)     /* CPU-0 なら */
{     (x)     }                             /* cache test */
else if(cpu_num=1)              (chain)     /* CPU-1 なら */
{     (y)     }                             /* cache test */
else if(cpu_num=2)              (chain)     /* CPU-2 なら */
{     (y)     }                             /* cache test */
else                            (chain)     /* CPU-3 */
{     (y)     }                             /* cache test */
```

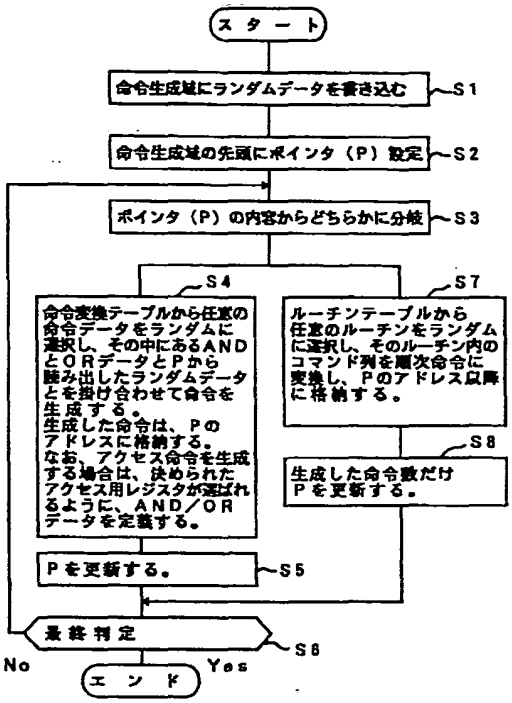
【図12】

例8の説明図



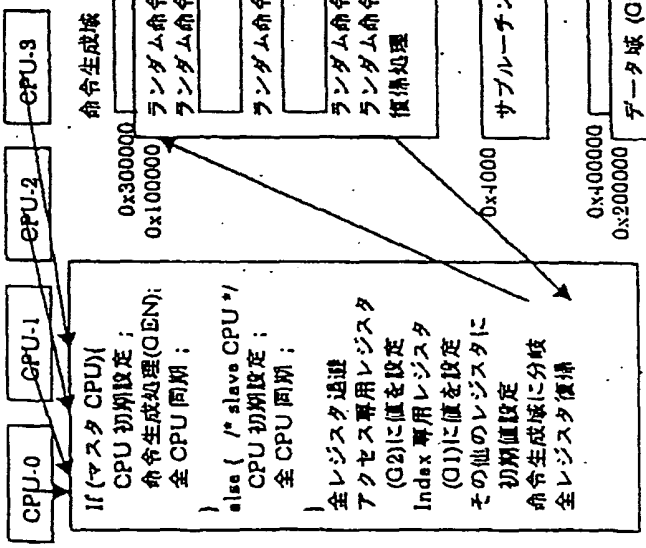
【図14】

ランダム命令列生成処理フローチャート

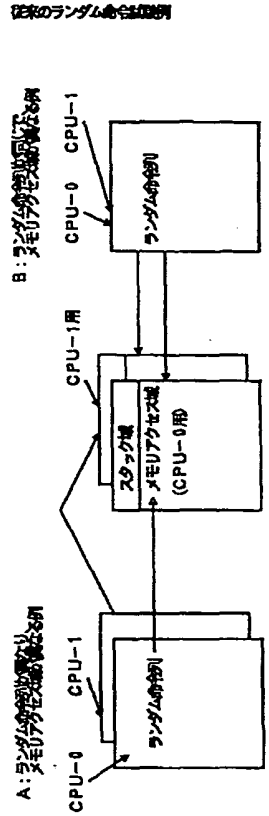


【図13】

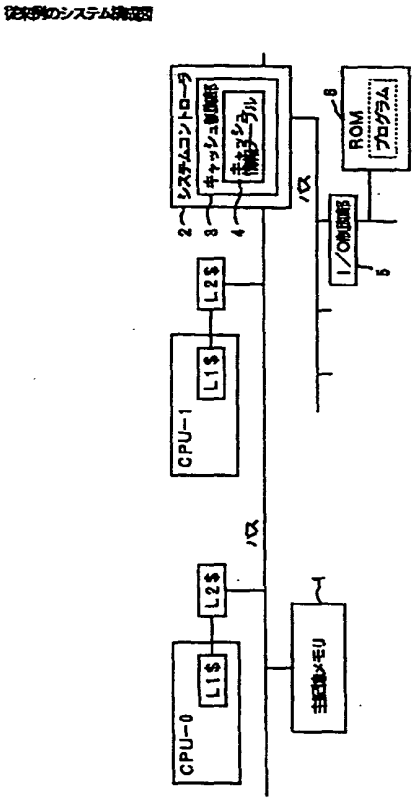
試験例の説明図



【図16】



【図15】



フロントページの続き

(51)Int.Cl.[7]		識別記号	FI		テーマコード(参考)
G06F	12/08	320	G06F	12/08	320
	12/12			12/12	A
	12/16	330		12/16	330B
	13/00	301		13/00	301T

Fターム(参考) 5B005 JJ01 KK12 LL11 MM05 MM12
PP12 RR02 UU32 VV22
5B018 GA03 HA32 MA03 QA13 RA11
5B045 BB12 DD12 JJ02
5B048 AA17 AA19 FF03
5B083 AA08 BB08 CE01 EE02